Pervasive and Mobile Computing ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

Fast track article

Social-aware hybrid mobile offloading

Huber Flores [a,*], Rajesh Sharma [b], Denzil Ferreira [a], Vassilis Kostakos [a], Jukka Manner [c], Sasu Tarkoma [d], Pan Hui [e], Yong Li [f]

[a] University of Oulu, Finland
[b] University of Bologna, Italy
[c] Aalto University, Finland
[d] University of Helsinki, Finland
[e] The Hong Kong University of Science and Technology, Hong Kong
[f] Tsinghua University, China

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
Mobile cloud
Cloudlet
Fog computing
Edge computing
Code offload

ABSTRACT

Mobile offloading is a promising technique to aid the constrained resources of a mobile device. By offloading a computational task, a device can save energy and increase the performance of the mobile applications. Unfortunately, in existing offloading systems, the opportunistic moments to offload a task are often sporadic and short-lived. We overcome this problem by proposing a social-aware hybrid offloading system (HyMobi), which increases the spectrum of offloading opportunities. As a mobile device is always co-located to at least one source of network infrastructure throughout of the day, by merging cloudlet, device-to-device and remote cloud offloading, we increase the availability of offloading support. Integrating these systems is not trivial. In order to keep such coupling, a strong social catalyst is required to foster user's participation and collaboration. Thus, we equip our system with an incentive mechanism based on credit and reputation, which exploits users' social aspects to create offload communities. We evaluate our system under controlled and in-the-wild scenarios. With credit, it is possible for a device to create opportunistic moments based on user's present need. As a result, we extended the widely used opportunistic model with a long-term perspective that significantly improves the offloading process and encourages unsupervised offloading adoption in the wild.

1. Introduction

Nowadays, mobile phones are used for more than making phone calls or sending texts thanks to the availability of countless mobile applications. However, these devices' mobility is constrained by battery life [1] and thus their usage is hampered. This is more critical when a user does not have access to a source for recharging the battery [2]. In such situations, as the battery life approaches its lowest energy levels, the user may experience temporary cognitive demands, such as frustration and anxiety, among others [3,4].

Researchers have developed multiple approaches to improve smartphones' battery life [5]. Computational offloading or cyber-foraging [6] is one of the most promising ways of extending the battery life as it deems to reduce the processing

* Corresponding author.
E-mail addresses: huber.flores@oulu.fi (H. Flores), rajesh.sharma@unibo.it (R. Sharma), denzil.ferreira@oulu.fi (D. Ferreira), vassilis.kostakos@oulu.fi (V. Kostakos), jukka.manner@aalto.fi (J. Manner), sasu.tarkoma@helsinki.fi (S. Tarkoma), panhui@cse.ust.hk (P. Hui), liyong07@tsinghua.edu.cn (Y. Li).

http://dx.doi.org/10.1016/j.pmcj.2016.09.014
1574-1192/

effort of applications running on the device in an opportunistic manner [7–14]. Simply put, computational offloading is a technique where a resource constrained device, e.g., CPU, battery, storage, outsources the processing of a task to a more powerful machine. In this process, the device weighs during runtime the effort to execute an application and calculates whether the cost of outsourcing a task from the application is less than the actual effort to process the task on its own. The cost of outsourcing the task is calculated by taking into consideration multiple parameters of the system [15], e.g., network latency, processing intensity of the code, surrogate capabilities, among others.

Computational offloading systems can be categorized into three different classes, namely (i) cloudlets [16], (ii) remote cloud [5] and (iii) device-to-device (D2D) [17]. Each system defines a particular opportunistic criteria to estimate the effort to offload. A mobile device that uses an offloading system, detects opportunities to offload when an application is executed. Thus, the augmentation of the mobile resources with external infrastructure is temporal as long as the criteria is fulfilled. By outsourcing a task, overall the mobile device consumes less resources, and in some cases, even the response time of the application is accelerated [5,18].

While different systems deal with the temporal resource augmentation of the mobile device in specific ways, the opportunistic moments provided by each offloading system are sporadic as each criteria need to meet many requirements, which change drastically during runtime. Since offloading support in a continuous manner is desirable to aid the mobile device with its processing, strategies to augment the spectrum of opportunistic moments are needed. Thus, in this paper, we tackle this challenge by proposing a hybrid system which integrates and unifies all the different types of offloading systems. The goal of our hybrid system is to increase the availability of offloading support for a mobile device.

Naturally, a hybrid system requires a cohesion catalyst to keep such union. Thus, our system exploits users' social aspects in order to achieve the unification of all the offloading systems into one. As a proof-of-concept of our hybrid system, we design and develop a framework namely *HyMobi*, which allows a mobile application to interoperate between offloading systems. However, it is a complex process to rely on user's participation and collaboration for maintaining an offloading system, mainly because using the devices of other users implies reducing their battery life. It has been demonstrated that the battery life of a smartphone has a personal yet quantifiable value [4]. Thus, to overcome this problem, *HyMobi* uses resource sharing incentives based on credit in order to lease and acquire offloading support.

The utilization of credit allows a device to create opportunistic moments to offload based on user's needs at any given time. In other words, *HyMobi* extends the short-term opportunistic view of common offloading systems in order to include a long-term opportunistic stance. *HyMobi* also uses the incentive mechanism to introduce reputation. This reputation along with the history information about detected infrastructure (i.e., stability), allows the *HyMobi* to identify spontaneous communities, which are used to automate the offloading process without user's intervention (*community offloading*). A community depicts a group of devices, which can be trusted to support offloading. Traditionally, existing offloading systems induce user's individualism, i.e., user's main goal is to obtain an on-site benefit for the mobile device, without necessarily allowing access to their own resources. In contrast, our hybrid system encourages collectivism, which means that a pool of computational resources is available for a device by relying on all the possible infrastructure that can be detected in the surrounding or in a remote location, e.g., smartphone, smartwatch, smartglasses, laptop, tablet, cloud server, and others.

To achieve this, *HyMobi* handles multiple technical and social challenges, summarized as follows:

- We develop *HyMobi*, which integrates cloudlet, remote cloud and device-to-device (D2D). Moreover, *HyMobi* extrapolates characteristics from a super-peer based system in order to introduce community roles and awareness in the hybrid system. This allows *HyMobi* to build a community with considerable ease.
- We design *HyMobi* with an incentive mechanism based on credit and reputation, depicted by points. A peer (i.e., user) gains points when it is contributing its computational resources to other peers' requests, e.g., by contributing resources, remaining in a certain location for a long time, pre-caching some operations, and others; and looses points when consuming resources of the community pool. The main benefit of our mechanism is to foster the users to lease the resources of their smartphones as open commodity that may be acquired by others. Moreover, we demonstrate that our proposed incentive mechanism based on credit is more effective to deal with selfishness of the peers when compared with other work in the field, e.g., altruism.
- Through a real-world deployment, we demonstrate that by adopting *HyMobi*, it is possible to improve the opportunistic ratio of computational infrastructure that is proximal to the mobile user. This suggests that the mobile device has higher opportunities to make its battery last longer with an external infrastructure. By relying on *HyMobi*, a device creates opportunistic moments to offload based on cumulative credit.

The rest of the paper is organized as follows. Section 2 describes background information and related work. Section 3 presents our proposed social-aware hybrid system along with its system implementation. Section 4 evaluates the performance of our proposed approach. Finally, Section 5 concludes the paper and presents the future directions.

2. Related work

In this section, we first present work related to mobile offloading in Section 2.1. Then, in Section 2.2, we present various approaches that study and evaluate selfishness in decentralized networking and computation systems.

Please cite this article in press as: H. Flores, et al., Social-aware hybrid mobile offloading, Pervasive and Mobile Computing (2016), http://dx.doi.org/10.1016/j.pmcj.2016.09.014

2.1. Mobile offloading

Computational offloading is the opportunistic process that relies on external infrastructure to execute a computational task outsourced by a low-power device. In this process, the device is granted with the decision logic to detect resource-intensive tasks, such that in the presence of network communication, the device can estimate where the processing of the task will require less computational effort (internal or external), which saves the device's energy [5].

Moving a computational task from one device to another is not a trivial endeavor [5]. Network latency plays a critical role in deciding whether or not to offload a task. A stable and low latency is preferable rather than an intermittent latency in the communication [19,20]. Initially a cloudlet system was envisioned, in which infrastructure proximal to the device, e.g., rich and trusted nearby servers, base stations, aids with its computational processing [16,21]. A cloudlet system identified the proximity of the infrastructure to the device (low latency network) as a key factor to offload in order to avoid energy overheads caused by the communication and to keep a smooth perception of the user towards the app. However, the complexity of development and deployment is a drawback for its adaptation.

As a result, later work utilized the cloud infrastructure, e.g., Amazon EC2, Azure, and others, to aid the device [22,23]. This remote cloud system relies on code offloading to reduce the amount of data exchange in the communication. Since the code granularity introduces different level abstractions to manipulate a task, multiple frameworks implement different granularity levels. MAUI [9], ThinkAir [11], EMCO [15] and COSMOS [14] implemented offloading at method level. CloneCloud [10] and COMET [13] developed a framework that offloads at the thread level. Some other work focus on other granularity levels like Class [24], jobs [25,26], etc. However, since the latency in the communication can change abruptly, the opportunistic moments to offload in a remote cloud system are sporadic. In addition, remote cloud offloading is sensitive to the multiple parameters of the system (context of the device), which means that it is challenging to pinpoint an opportunistic moment to offload.

Consequently, in order to address the issues of remote cloud and cloudlet systems, most recent work rely on other smartphone devices that are carried around by other users (transient infrastructure) [17,27,28]. This proximity system is called device-to-device (D2D) and produces a low latency environment to offload. Moreover, its deployment is relatively easy as smartphones are equipped with similar runtime environments and different means for communication, e.g., Bluetooth, WiFi-Direct. It has been demonstrated that social relations [28], e.g., friendship [29–32], and social attitudes [27,17], e.g., altruism, tend to provide temporal offloading support in a D2D system. However, a D2D system is not self-sustainable. A D2D system is sensitive to collapse based on how the user perceives the system socially, e.g., motivation, benefits, costs, etc. Our previous work [4] has identified that the value that the user puts on battery life can be quantified and providing temporal support is not appealing for a user. As a result, the adoption of a D2D system is stuttering.

Nonetheless, we argue that a user can lease the resource of his/her devices as long as the leasing provides a benefit that can be accumulated in order to be used later to create opportunistic moments to offload based on the user's needs. (i.e., a long-term opportunistic perspective). Thus, unlike previous work, we propose to merge all the systems into a hybrid one, which also implements a credit mechanism that encourages a user to obtain a benefit from his/her device. This credit can be accumulated and used within the system to support the device of the user when required.

While there has been attempts to unify all the offloading systems by other works [33], those proposals still foster an individualistic behavior to the user, mainly because they focus on a short-term opportunistic resource leases. In contrast, our hybrid system implements an incentive approach based on credit to motivate the mobile users to participate actively and accumulate more credit. To achieve this, our hybrid system extrapolates characteristics of a super-peer based system in order to foster an offloading system that can be sustained by a community on the long run.

Lastly, to fully demonstrate the potential of our *HyMobi* framework, we provide a comparison with previous approaches. Table 1 provides a review of existing frameworks and their features. It provides information about the type of system that each framework adopts (offloading system), shows whether the framework considers some social aspects that can be exploited to turn a device into an open commodity that can be leased (social awareness), and shows whether the framework adopts a short-term (individual) or long-term (community) opportunistic perspective to provide offloading support. *HyMobi* is the first framework developed and designed to take into consideration the personal and quantifiable value of mobile devices' battery life. *HyMobi* is equipped with a credit and reputation mechanisms that is attractive for social participation and collaboration. Finally, the table shows the approach used by each framework to integrate different offloading systems (integration). In contrast to other work, *HyMobi* extrapolates features from a super-peer based model in order to introduce community roles, so that the system can be sustained by the users. This also improves scalability as the available infrastructure is registered or unregistered in the super-peers dynamically.

2.2. Self-organizing networks: selfishness

A self-organizing network is formed by a set of dynamic mobile nodes without relying on any established infrastructure [34]. Due to the lack of infrastructure, the nodes must perform network functions by themselves in a decentralized manner, e.g., packet forwarding and routing.

In decentralized settings, selfish behavior of nodes can affect the efficiency of the system. Examples of such systems include Mobile ad-hoc networks (MANET) [35], wireless networks [36], Massive-Multiplayer online games (MMOG) [37],

Please cite this article in press as: H. Flores, et al., Social-aware hybrid mobile offloading, Pervasive and Mobile Computing (2016), http://dx.doi.org/10.1016/j.pmcj.2016.09.014

Table 1
Qualitative comparison of the offloading frameworks.

Framework	Offloading system			Features		
Name	Cloudlet	Cloud	D2D	Social awareness	Offloading support	Integration
MAUI [9]	–	x	–	None	Individual	Static
ThinkAir [11]	–	x	–	None	Individual	Static
CloneCloud [10]	–	x	–	None	Individual	Static
Odessa [25]	–	x	–	None	Individual	Static
EMCO [12,5]	–	x	–	None	Individual	Static
COMET [13]	–	x	–	None	Individual	Static
Cloudlet variations [16,21]	x	x	–	None	Individual	Static
Phone2Cloud [24]	–	x	–	None	Individual	Static
COSMOS [14]	–	x	–	None	Individual	Static
Serendipity [17]	–	–	x	Altruism	Individual	Static
FemtoClouds, MDC [29,31]	x	x	x	Friendship and acquaintances (history connections)	Individual	Static
HyMobi	x	x	x	Incentives and reputation	Community	Dynamic (super-peer based)

Peer-to-peer (P2P) systems [38,39] and systems based on decentralized architectures such as Decentralized Online Social networks (DOSN) [40].

2.2.1. MANET and wireless systems

Mobile ad-hoc networks (MANET) is one of the closely related domains which suffers from the selfishness of the nodes [35,41]. A two level network-layer acknowledgment-based approach was presented in [42] to track and alleviate the misbehaving nodes from the system. An incentive based cooperative mechanism was proposed in [43]. In a different work, a reputation based incentive mechanism was proposed in [44]. Wireless networks also suffers from the selfishness behavior. They also need proper mechanism to detect selfish nodes [45]. Various game theoretic models have been used to evaluate schemes such as incentive based scheme [36] or tit-for-tat [46]. Selfishness also is explored in DTN (Delay Tolerant Networks). Multiple mechanisms were proposed to overcome the selfish behaviors of nodes that affect the performance of DTN multicast [47]. In comparison with these studies, we propose an incentive based solution, in which the system is sustained by the users in the community.

2.2.2. P2P and related systems

The problem of selfishness is natural to appear in other kinds of decentralized systems. In Peer-to-peer (P2P) systems there is a high tendency of nodes becoming leechers and with a few contributors [39]. The selfish nodes in P2P system often indulge in selfish routing [38]. To overcome this mean behavior, incentive based mechanisms have been proposed for P2P systems [48–50].

The problem of selfishness was also investigated in systems built on top of P2P or related technologies such as decentralized online social networks (DOSN) [40]. The DOSN cannot be successful if a node just relies on friends [51]. A game theoretic model was used to study the feasibility of DOSN [52]. The study showed that there is a high possibility of rich resources colluding among each other for individual good performance. A super-peers based altruistic system was proposed to make the system successful [53]. Our approach is partially inspired from [53], where we take care of the new joinees in the systems. On top of it, we propose an incentive mechanism to introduce fairness in the system.

2.2.3. Incentive schemes

To introduce fairness in distributed and decentralized systems, incentive based mechanisms are needed [54,55]. These schemes can be divided into two main categories, namely (i) reputation based schemes and (ii) credit based schemes.

In reputation based schemes, nodes provide feedback about the services they receive. For example, users can monitor their neighbors and use their forwarding nature to calculate reputation [44]. This naming and shaming approach will not work well in an environment where users are often mobiles.

In credit based schemes, a virtual currency is used for rewarding the user (for example, [56]). This scheme ensures that selfish nodes run out of their credits eventually forcing them to either leave or participate in the system. Users have to use their currency to take services from others and the users which provide the services earn the credits. In our system, credit for a device is obtained from modeling the resources of that device as an open commodity that can be leased.

3. Hybrid system overview

In this section, we first describe the overall hybrid system (3.1). Next we describe the entities (Section 3.2) and services (Section 3.3) in detail. Later, we briefly discuss our incentive mechanism (Section 3.2.1) before describing the approach of connection establishment among nodes.

Please cite this article in press as: H. Flores, et al., Social-aware hybrid mobile offloading, Pervasive and Mobile Computing (2016), http://dx.doi.org/10.1016/j.pmcj.2016.09.014

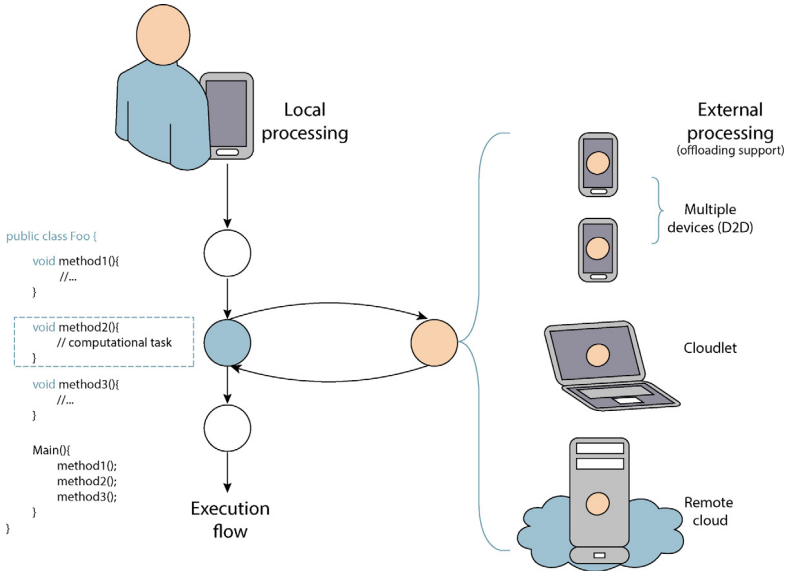


Fig. 1. Hybrid computational offloading system.

3.1. Social-aware system

The proposed hybrid system for computational offloading is shown in Fig. 1. In this system, a device can outsource a computational task to any available surrogate, e.g., cloudlet, remote cloud or D2D. An offloading request is created between devices based on a credit relation. This credit relation implies that a user can lease and acquire computational resources to and from other users, respectively. When this relation is established, the devices can create opportunistic moments to move the processing of a computational task among the interconnected devices. Naturally, the creation of such opportunistic moment depends on the credit that a particular user has accumulated. Credit is mainly accumulated from leasing, and consumed, when acquiring resources. Other criteria to gain credit can be defined at framework level based on the role of each node.

Since a user may own multiple devices, for simplicity, we assume that the credit is assigned at a user profile level and not at particular device (Fig. 2(a)). Thus, a user can use the credits it gains for any device that he/she owns, e.g., smartwatch, tablet, glasses, laptop, etc. When a mobile device attempts to rely on external infrastructure (which the user does not own), the device notifies the user about the need to acquire offloading support. The user then has to select explicitly from whom it is going to acquire the offloading support. Since each user profile has collected reputation, the user can rely on that information to select the best fit (trustful).

Naturally, notifying each time the device requires offloading is disturbing for the user, thus the system implements a criteria that uses the reputation, history of connections and stability of the external infrastructure to create offloading communities in which the offloading process is automated (Fig. 2(b) and (c)). This means that the credit flows between the users without users' intervention, and the reputation is used to determine how good is the service provided by a user. The history connection is used to verify previous experiences by acquiring support from a user and whether the connection was successful or not. The stability of the infrastructure is also taken into consideration to identify how frequent and how long a user is perceived as infrastructure in proximity (Fig. 2(d)). Moreover, these three mechanisms altogether are used to decide whether a user is trustful or not. Thus, an *offloading community* is formed among users that fulfill the community criteria.

3.2. Entities

The system[1] consists of various nodes, where N represents the node's set. The nodes could be static or mobile. Examples of static nodes include cloud-based servers or cloudlets, and users with mobile devices are examples of mobile nodes. The status st of each node $n \in N$, represented by n_{st} could either be a peer p or super-peer sp. That is $n_{st} = p$ or sp. We define these two entities as following:

1. **Peer**: Every node in the system by default is called peer. This includes Cloudlets, any remote server providing system services or any mobile device.

[1] The term system signifies the framework for *HyMobi*.

Please cite this article in press as: H. Flores, et al., Social-aware hybrid mobile offloading, Pervasive and Mobile Computing (2016), http://dx.doi.org/10.1016/j.pmcj.2016.09.014

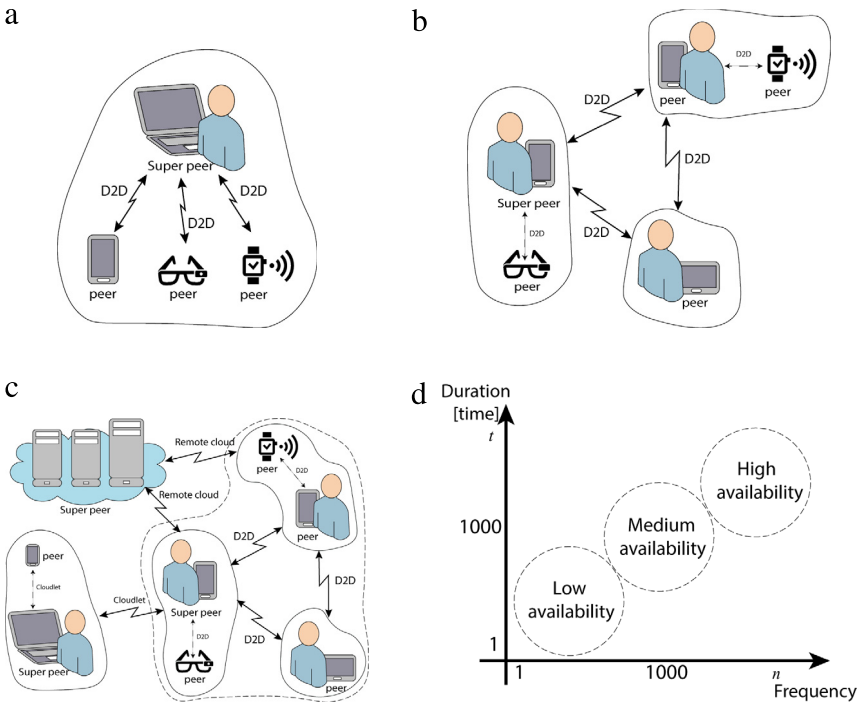


Fig. 2. Social-awareness and infrastructure integration. (a) A private community formed by the devices that a user owns, (b) and (c) communities between multiple users, (d) stability of the external infrastructure.

2. **Super-peers**: These entities provide system level services so that the overall system can sustain. Cloudlets and remote servers are naturally super-peers as system services will be running over these servers. Any peer can become a super-peer if it is running a particular system level service using its resources. Roles of super-peers include facilitators of providing information about users in a particular locality and mediator in case of conflicts between resource provider and resource consumer.

Communities: A community is formed when the exchange of credit between peers is automated. As described before, this is ensured based on the community criteria of the system. For instance, consider the following two cases.

1. During particular hours, in the weekdays users tend to work in a same workplace or study in the same department/institute. Generally, they encounter the same people during a specific time period. Thus, this leads to the notion of temporal community among the peers which are regularly present at a specific time period in specific locations. Thus, these peers can easily meet the requirements of reputation, history connection and stability of the community criteria.
2. In contrast to the above case, a traveler may encounter plenty of offloading support during a trip. However, this infrastructure does not meet the requirements to form a community. Thus, the user relies solely on the reputation of the peers, that are found in order to decide whether or not to request offloading support from them. This process is explicit requested by the user initially when new peers are discovered.

3.2.1. Incentive scheme

To make the system sustainable, we introduce an incentive mechanism. Our incentive approach is a hybrid model incorporating both reputations as well as a credits. Reputation mechanism is put in place for users to select quality providers. To make the system successful in realistic scenarios where users exhibit bounded rationality, we introduce a credit based mechanism. Each user (or node) n_i has a credit value of n_{Cr_i}. Considering a node seeking resources (called consumer), and denoted by n_c, a node ready to provide service denoted by n_p, and the super-peer facilitating the agreement between two, represented by n_{sp}. The principle behind our credit based system is based on following rules:

1. Consumers: the credits (δ) will be deducted from the user for using other users' resources. Thus, $Cr_{n_c} = Cr_{n_c} - \delta$
2. Providers: the credits will be transferred to the user providing the resources. Thus, $Cr_{n_p} = Cr_{n_p} + \delta$
3. Facilitator: some share of credits will also be added to the super-peer account. Thus, $Cr_{n_{sp}} = Cr_{n_{sp}} + \phi$ where $\delta > \phi$.
4. For running system level services, super-peers receive double credits for providing the resources compared to providing resources to another resource seeking node. This strategy of double earning is kept in place to motivate higher number of nodes to act as super-peers and thus, avoiding the system to collapse when super-peers, such as remote cloud or cloudlet are not reachable to any peer. Naturally, the overall payment system can be adjusted to any scale as long as it is the same for all nodes of the system.

Please cite this article in press as: H. Flores, et al., Social-aware hybrid mobile offloading, Pervasive and Mobile Computing (2016), http://dx.doi.org/10.1016/j.pmcj.2016.09.014

In addition, along with credit based mechanism, the system also has a reputation system which helps in deciding the nodes about the service providers. After every transaction, each service provider and consumer has an option to provide feedback to each other. The goal of the reputation mechanism is to allow a super-peer to gain more visibility to lease resources to other peers. Moreover, this introduces quality control into the system, such that a node seeking resources trustfully obtains a benefit (energy or response time) from the offloading process. To calculate reputation, we introduce a simple point based reputation (which is different from credit based incentive mechanism). Every time a consumer requests for the list of service providers, the system calculates the reputation based on the following formula:

$$Rep(n_{p_t}) = \begin{cases} 0 & \text{if } |ts| = 0 \\ Rep(n_{p_{ts=1}}) & \text{if } |ts| = 1 \\ \alpha\left(\left(\sum_{ts=1}^{ts=\theta} Rep(n_{p_{ts}})\right)\Big/\theta\right) + \beta\left(\left(\sum_{ts=\theta+1}^{ts=\sigma} Rep(n_{p_{ts}})\right)\Big/(\sigma-\theta)\right) & \text{if } |ts| > 1. \end{cases} \quad (1)$$

Eq. (1) denotes the calculation of reputation (Rep) for a service provider n_p at the time t. If a particular service provider has no transaction history ($|ts| = 0$, where ts denotes the transaction history) then its reputation is considered zero. In case there is a single transaction ($|ts| = 1$) then the reputation of service provider n_p is returned as it is. If the service provider has provided its services for more than one time, then the reputation of the service provider n_p is average of all the reputation feedback given by all the users in the past. However, recent transactions have more value than older ones. The term *recent* is subjective and can be adjusted by the system based on the request of the consumer. In Eq. (1), θ signifies the recent transactions and σ the total transactions for n_p. Further, recent and older transactions can be tuned using the parameters α and β, where $\alpha + \beta = 1$. It is to be noted that the higher the reputation value of a node, the more trustworthy it is among the users seeking resources.

Through credit based scheme users will be motivated to earn credits by contributing to the systems. Additionally, we embed the reputation mechanism that helps in reinforcing quality service by peers for each other in the system.

3.3. Services

We envisioned various services which must be running 24/7 for users. Some of the examples of these services include:

1. **Registration**: At regular intervals, users registers with at least one super-peer. During the registration process, a user provides its geolocation, which helps the super-peer in calculating the proximity of the user.
2. **Gateway**: The gateway service provides connectivity of users to the facilitators of the system which includes servers and super-peers.
3. **Find super-peers list**: Using super-peers list, a user first connects to the super-peers. The role of super-peer is to provide locality based information to the users. The system returns a list of tuples consisting of super-peer id, proximity of the super-peer to the user seeking resources ($dist$) at time t, and the reputation of super-peers (Rep). Formally, the returned list consisting of M super-peers, can be represented by $\langle\{sp_1,\ dist_1,\ Rep_{sp_1}\},\ \{sp_2,\ dist_2,\ Rep_{sp_2}\}, \ldots\ \{sp_M,\ dist_M,\ Rep_{sp_M}\}\rangle$.
4. **Find user list**: Once the users connect to the facilitators (super-peer in this case), it is provided with a list of users in a particular location which are offering their resources. For any particular time t this list provides the condition of users' resources (represented by Cn). The resource condition includes status of the resource and quality of the resource. By status, we mean for how long it can provide the service, for example, in next 1 h only. Examples of the quality of resources include Samsung Galaxy 6, 5, 4, and 3. The list also provides the reputation of all the nodes in the list. The returned list consisting of K peers and can be represented by $\langle\{p_1,\ dist_1,\ cn_1,\ Rep_{p_1}\},\ \{p_2,\ dist_2,\ cn_2,\ Rep_{p_2}\}, \ldots.\{p_K,\ dist_K,\ cn_K,\ Rep_{p_K}\}\rangle$.
5. **Reputation and credit propagation service**: After the completion of each offloading task, using push technologies [57] super-peers receive updated information about reputation and credits of the interacting peers. Furthermore, using this service, super-peers propagate the information about reputation and credit of users with each other at regular intervals. We assume that this service is secure enough from malicious attacks. Topics related to security such as dealing with malicious users and information theft are out of scope of this work.

3.3.1. Approach

The protocol for connection between resource requester and resource provider using super-peer is as follows:

1. Each resource provider gets enlisted with at least one super-peer and provides its location during registration.
2. When a user is looking for resources in a particular location. It contacts the gateway service, which returns the list of super-peers based on proximity as well as of reputation. A resource seeker can always prioritize his option while seeking super-peer list. That is if it wants a list based on proximity or reputation. A user can then select a super-peer based on reputation or proximity or both. The selected super-peer provides the list of resource providers for the resource seeker. Each resource provider is also enlisted with the quality of resources and the time units for which it can provide the services with the mentioned quality. A user then selects the resource providing peer from the list based various parameters that is the quality of the service, time units it desires, reputation and proximity from the service provider.

Please cite this article in press as: H. Flores, et al., Social-aware hybrid mobile offloading, Pervasive and Mobile Computing (2016), http://dx.doi.org/10.1016/j.pmcj.2016.09.014

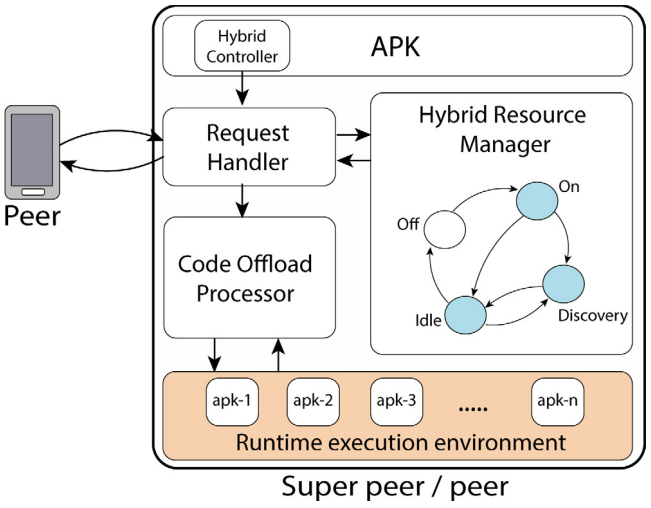


Fig. 3. Architecture of the social-aware hybrid offloading framework (HyMobi).

3. super-peer introduces the resource provider with the resource requester in terms of connection establishment. After an agreement between the two parties has been done, super-peer notes the points to be deducted from the requester and to be rewarded to the resource provider. In lieu of the services, the super-peer is awarded x% (ϕ) of these points.
4. After completion of the job, super-peer gets a notification. On notification, super-peer makes the actual transactions of credits for two parties in terms of addition and deduction of credit points.

To care of new joinees, each of them is provided with fixed credit points and for a fixed period of time. This forces the new joinees to start contributing from the beginning to earn credits. Alternatively, a new joinee can run system level services as it helps in earning credits faster compared to allocating its own resources for resource seekers. This mechanism induces more super-peers in the system and thus helps in the sustenance of the system.

4. System implementation

In this section, we provide detailed information about the implementation of *HyMobi*[2] system. Fig. 3 presents the overall architecture and provides relation between various components. Each device is equipped with peer as well as super-peer capabilities. As a device can change dynamically to perform different roles, e.g., peer consumer, peer leaser, super-peer, each component of the architecture provides a functionality based on the role of the device. Each component is discussed as follows:

Runtime execution environment (REE): It contains the necessary compiler to execute the code of an APK file. In the case of mobile devices, e.g., smartwatch, smartphone, tablets, etc., by default, these devices are already equipped with such execution environment. In the case of a cloudlet or a cloud server, we equipped them with a Dalvik-x86 compiler [5], which was extracted from the version of Android for x86 machines. Dalvik-x86 is lighter when compared with Android-x86 as it just contains the necessary means to execute code. Thus, we rely on Dalvik-x86 to grant a peer with a homogeneous environment of execution. Dalvik-x86 implements an executable script wrapper at the core of the libraries that boot the compiler. The wrapper provides an interface to push bytecode for execution as a system process in the host OS.

Code offload processor (COP): This component invokes the code from the details which are defined in the code offloading request. Since offloading a task involves reconstruction of code in the target peer leaser, thus, the leaser needs to have the same application installed as the consumer [5]. As a result, in order to invoke the code, this component loads the necessary classes from the available APKs which are deployed in the *REE*. Once the code is executed, the result of the invocation is packed and sent back to the peer consumer, such that the peer consumer can update its application state.

Hybrid resource manager (HRM): It wraps the available network interfaces of the peer into a state machine of four generic states (On, Off, Idle and Discovery). Each state controls a particular functionality of the network interface, e.g., the *Discovery* state induces the interface to search for proximal devices. The utilization of state machine allows a peer to use the same network controller to manage different types of communication. Thus, a peer can change from the communication channel during runtime. We implemented offloading via WiFi, WiFi-Direct and Bluetooth. Since WiFi and WiFi-Direct use the same physical interface, the same implementation can be re-used for both. However, in the case of Bluetooth, the communication sockets are different, and thus, a different implementation has been put in place.

2 https://github.com/mobile-cloud-computing/HybridComputationalOffloading.

Please cite this article in press as: H. Flores, et al., Social-aware hybrid mobile offloading, Pervasive and Mobile Computing (2016), http://dx.doi.org/10.1016/j.pmcj.2016.09.014

Request handler (RH): It is the gateway where a request is processed based on its type, where the type of the request depends on the role of the peer. In a super-peer role, discovery and transaction requests are handled. When a discovery request is received, the *RH* responds to a peer with a list of peers from the information provided by the *HRM*. Similarly, when a transaction request is received, the super-peer is in charge of (i) handling the transfer of credits between the peers and (ii) assignment of reputation based on user's feedback. At the end of each transaction, the super-peer uses the GCM (Google Cloud Messaging), which is a push mechanism to propagate updates about reputation and credit to other peers. The mechanism can also communicate via more generic mechanisms like XMPP [57] in order to send updates to any device. In a consumer role, an opportunistic moment is created with credit. Thus, an offloading request is created. The *RH* retrieves from the *HRM* the list of available peers to offload (if available). Otherwise, the peer sends a discovery request to the super-peer. Once the consumer decides the target for leasing the code, the offloading request is sent to the leaser. Lastly, after the result of the offloading request is received by the consumer, the consumer sends a transaction request to the super-peer in order to exchange the credit and give feedback. In a leaser role, the *RH* passes the request (which contains the details of the code) to *COP*, such that *COP* can execute the request using its computational resources.

Hybrid controller (HC): This component captures the execution details of a computational task during runtime at the method level, e.g., name of the method, parameters, type of the method, etc. Inspired by [5,12], the instrumentation of code with this controller is automatic done using a graphical interface tool. Similarly to Continuous Integration (CI) frameworks, e.g., Heroku, the developer just has to provide the link of the repository where the application is located. The tool parses the source code and transforms those potential methods that are candidate to offload [10]. For each method, the transformation consists in creating an offloading method that uses Java reflection to capture the runtime details of the original method. Moreover, in this transformation, the prefix *local* is appended to the name of the original method while the offloading method takes its original name. The instrumentation of the hybrid controller is as follows.

```
public class BattleGame extends HybridRemotable {

    //Original method
    public StringBuilder localloadGameSprite3D(String filename){
       //routine
       return model;
    }

    //Offloading method
    public StringBuilder loadGameSprite3D(String filename) {
       Method toExecute;
       Class<?>[] paramTypes = {String.class};
       Object[] paramValues = {filename};
       StringBuilder result = null;

       try{
          toExecute = this.getClass().getDeclaredMethod("localloadGameSprite3D", paramTypes);
          Vector results = getHybridController().execute(toExecute,
                                                         paramValues,
                                                         this,this.getClass());
          if (results != null) {
              result = (StringBuilder)results.get(0);
          }else{
              result = localloadGameSprite3D(filename);
          }
       } catch (SecurityException se){}
       catch (NoSuchMethodException ns){}
       catch (Throwable th){}

       return result;
    }
}
```

The hybrid controller is active when the mobile application is being used by the user (peer consumer). In order to offload a task, the *HC* passes the offloading request to *RH*.

5. Evaluation and analysis

In this section, we present the evaluation of *HyMobi*. We evaluate and analyze two different aspects of the framework, (i) mobile application performance and energy consumption (lab testbed), (ii) infrastructure awareness in a social-aware environment (real testbed). In our experiments, we used the devices described in Table 2. We also used a SWR50 SmartWatch 3, laptop computer (Intel Core i3, 2.3 GHz, 4 GB of memory) and a general purpose instance m3.large of Amazon EC2 (Ireland region/eu-west-1). To measure the energy consumed by the mobile in our offloading experiments, we relied on the appliance, namely Mobile Device Power Monitor.[3]

5.1. Performance and energy consumption

Setup and methodology:—The goal of this experiment is to evaluate the gains in performance obtained by the mobile applications that implement *HyMobi*. We analyze the effect of offloading a computational task to different types of peers (Cloudlet, Remote cloud and D2D). We select a *QuickSort* algorithm as a candidate offloading routine. *QuickSort* is considered as it is a common logic routine, whose implementation can be found in many applications, such as game animations, text

[3] http://www.msoon.com/LabEquipment/PowerMonitor/.

Table 2
Technical specification of the mobiles used in the social-aware experiment.

Id	Units, device, and mobile platform	CPU (GHz)	RAM (GB)
p0	1, Samsung Galaxy S3, Android	Quad-core 1.4	1
p1	1, Motorola Droid Turbo, Android	Quad-core 2.7	3
p2	1, Google Nexus, Android	Dual-core 1.2	1
p3	1, Google Nexus, Android	Dual-core 1.2	1
p4	1, Sony Xperia Z3, Android	Quad-core 2.5	3
p5	1, Samsung Galaxy Note 5, Android	Quad-core 1.5	4
p6	1, Alcatel X1, Android	Quad-core 1.4	2
p7	1, Samsung Galaxy S2, Android	Quad-core 1.2	1
p8	1, Samsung Galaxy Alfa, Android	Quad-core 1.8	2
p9	1, Samsung Galaxy S6 Edge, Android	Dual-core 1.5	4

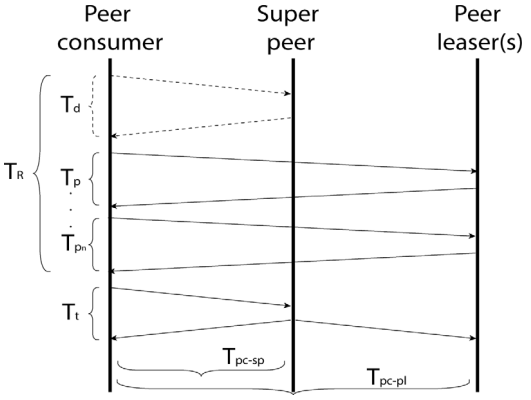


Fig. 4. Timestamps and processing procedure of an offloading request in HyMobi.

translation, recommendation systems, etc. In order to cover different cases of non-deterministic code execution, multiple variations of *QuickSort* based on input variability (size of the array) are considered. *QuickSort* becomes gradually resource intensive as the size of the elements to sort increases. Moreover, the algorithm can be divided into several recursive subtasks, which facilitates the process of splitting the execution of a task in order to be offloaded to multiple peers (code parallelization).

Results:—Since *HyMobi* introduces the credit and feedback mechanisms, the steps to offload a computational task differs from a classical offloading framework. Fig. 4 shows the offloading process of a peer that uses HyMobi. From the Figure, it is possible to observe the different times that influence the response time of the code execution. The total response time of an offloaded task is given by $T_R = T_d + T_p$, where T_d is the time that takes to discover and decide the peer(s) to offload, and T_p is the time that takes to process the computational task by the peer(s). Notice that once a peer consumer gets the list of peers from the super-peer, it can avoid T_d as long as the temporal span between the creation of offloading requests is short or the peer consumer remains in a stable region of the community [20]. T_p is influenced by the effort of establishing the communication between peers. Thus, $T_p = T_{pc-sp} + T_{sp-pl}$, where T_{pc-sp} is the time that takes to send a request from the peer consumer to the super-peer, and T_{pc-pl} is the time that takes to send the actual code request from the peer consumer to the peer leaser. In both cases, the request time includes the round trip time (RTT). Moreover, T_p also depends on the number of peers involved in processing the task. Thus, when a task is offloaded to more than one peer, T_p changes to $\sum_{i=1}^{n} T_{p_i}$. Also, T_t is the time that takes to exchange the credit and give feedback. However, T_t do not influence the response time of the mobile application, which is perceived by the user.

Since the execution of the code also depends on the computational capabilities of the peer's resources, we measure the execution time of the *QuickSort* task in different peers. Fig. 5(a) shows the results. We can observe that the devices with a low pool of computational resources, such as the smartwatch or smartphones, require long time to process the task. In contrast, powerful devices such as a cloudlet or a cloud server can process the task with considerable ease.

We measure the times that influence the response time of an offloading request. Fig. 5(b) shows a peer consumer (smartwatch) requesting to a super-peer (laptop cloudlet) the list of available peer leasers ($T_d \approx 280$ ms), and then offloading the task (sort elements $=$ 20k) to the selected peer leaser (i9300) ($T_p \approx 7.5$ s). After the processed task is received, the consumer initiates the credit exchange ($T_t \approx 550$ ms). In this case, T_t is automatic without feedback. When feedback from the user is considered, an additional time is added to T_t, which is the time taken by a user for providing feedback about the services being availed.

Latency in the communication also influences the response time of the application. We measure the latency in the communication when offloading to different peers. Fig. 5(c) shows the results. We can observe from the results that the latency of cloudlet and D2D is very small (mean $\approx$ 110 ms, SD $\approx$ 30.9). In contrast, the latency of remote cloud is high

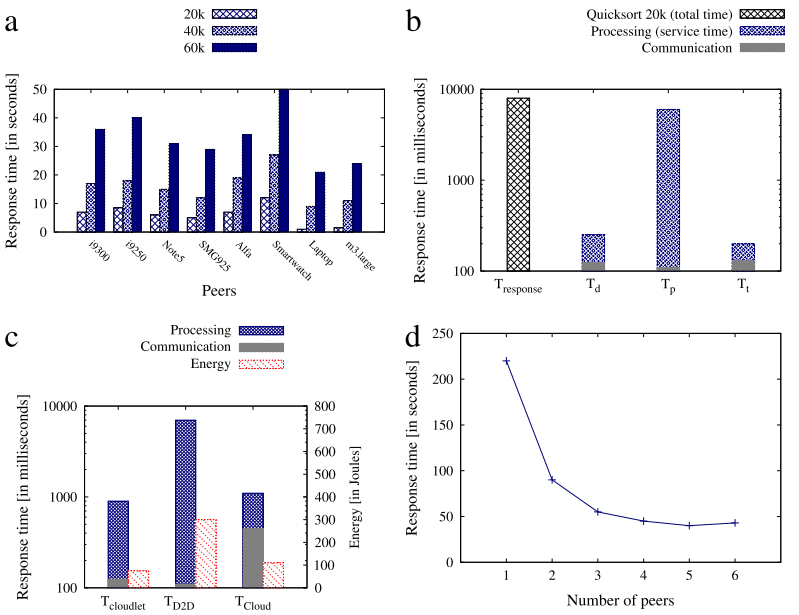


Fig. 5. Performance analysis of the offloading process. (a) Execution of *QuickSort* in different devices using different inputs, (b) times of an offloading request throughout the components of the system, (c) execution of a task in different types of peers, and (d) task execution divided into multiple peers.

(6x in this case) and can change abruptly. Fig. 5(c) also shows information about the energy consumed by the peer when offloading using a particular system. From the results, we can observe that it is power efficient to offload to a cloudlet or remote cloud as the task can be processed easily. In contrast, by using D2D, the peer consumer experiences some gains in energy, but the leaser experiences high energy drain.

We demonstrate how the response time of an application can be improved when a task is offloaded to multiple peers (sort elements = 200k). Fig. 5(d) shows the results. We can observe that the execution of the task can be accelerated when the peer parallelizes the task among peers. However, similar to remote cloud offloading [5], the parallelization of a task is just beneficial when the task to execute is highly resource intensive, which means that the task requires a lot of computational resources and longer execution time. Otherwise, it degrades performance. The considerations for task parallelization are out of the scope of this paper and will be addressed in future work.

5.2. Social infrastructure awareness

By relying on our framework *HyMobi*, we develop a mobile application called *Detector*.[4] The application searches for available peers to offload using three communications means, WiFi, WiFi-Direct and Bluetooth. In the case of WiFi-Direct and Bluetooth, the scanning happens every 15 min. We consider this interval to be enough as different peers have the application installed and during that time, multiple devices from different users will trigger the scanning routine to discover each other. In the case of WiFi, the applications just calculate the RTT to a remote server located in Amazon EC2 Frankfurt. The information of the peer discovery process is stored in a SQLite database and contains information in the key–value format, which includes the type of device, device's name, RTT, battery discharge rate, Bluetooth, WiFi-Direct and WiFi addresses.

The *Detector* application is hosted in Google app store (beta service in order to facilitate its deployment and distribution). The logic of the application is quite simple for the user, who just have to grant permission to the application to use the Bluetooth of his/her device. Once the permission is granted, the application can collect information in the background, which means that *Detector* does not interfere with the usage of the mobile device. In order to achieve this, the application implements a scheduling routine that activates the discovery process of each network interface. Each day the applications uploads the database to a remote server located in Amazon EC2.

Setup and methodology:—The goal of this experiment is to validate the enhancement in availability regarding the infrastructure proximal (offloading support) to the device. We recruit 15 participants to install the *Detector* application and collect daily information of each participant during one month experiment. From the 15 participants, we discard 5 as the data collected from those users did not contain enough information to re-construct the sensing process during each day. We use the traces collected from the participants to conduct an analysis of offloading support, community and credit simulation.

[4] https://github.com/mobile-cloud-computing/HybridComputationalOffloading.

Please cite this article in press as: H. Flores, et al., Social-aware hybrid mobile offloading, Pervasive and Mobile Computing (2016), http://dx.doi.org/10.1016/j.pmcj.2016.09.014

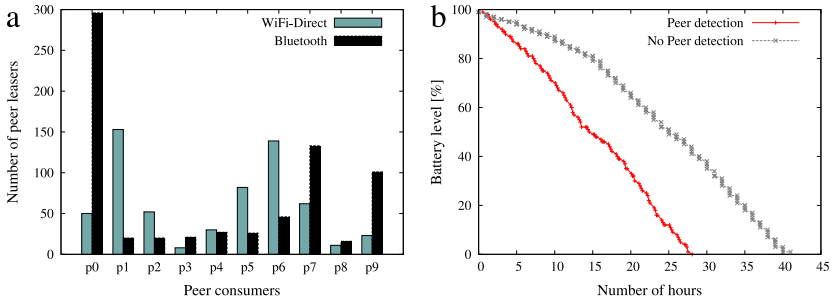


Fig. 6. Number of peers leasers detected per each peer consumer that installed the *Detector application* (b) energetic effort (battery life) required for a particular peer to detect available peers using *Detector*.

Results: community—Based on the data collected, we calculate the total number of leaser peers that were discovered by each participant. Fig. 6(a) shows the results for detection of peers via WiFi-Direct and Bluetooth. While some peers were able to discover a large amount of peers (p0, p1, p6, p7, and p9), some others just discover a few devices (p2, p3, p4 and p8). This is reasonable as different users have different daily life's routines and some users are more active than other (in terms of mobility). Thus, these results also provide insights about the mobility behavior of each user.

Naturally, the discovery process of the peers in proximity induces a high computational load in a device. This is important if we consider that a user gets used to a specific routine to charge his/her phone [1]. Thus, when a power-hungry application is installed in the user's device, it may cause disturbance as the routine of the user is affected. Fig. 6(b) compares the energy draining of a device (p3) when the peer detection functionality is active and inactive. From the results, we can observe that the peer detection consumes ≈12 h of battery life of the device. As a result, the addition of the super-peer in our model is reasonable as the super-peer is the only one affected by periodically monitoring the available infrastructure in proximity. Since the super-peer gets additional credit for this task, thus it has good motivation from the system to perform that role.

We proceed to determine the communities that were identified by the participants during the one month experiment. We calculate the stability of each peer discovered by estimating the frequency and duration in which each peer is addressable or not (visible or invisible). Fig. 7 presents the results of the analysis for each participant. From the results, we can clearly appreciate that there are different levels of intermittent behavior of the peers (power law). Peers with higher intermittent behavior are the most suitable to create communities in which the offloading process can be automated.

Results: availability— Figs. 8 and 9 shows the average number of peers discovered by each participant during each hour of the day via WiFi-Direct and Bluetooth, respectively. Additionally, we also provide information about battery discharge for each user in Fig. 10 in order to visualize the time periods that can benefit from the detection of proximal infrastructure. From the results, we can observe that WiFi-Direct provides the largest number of addressable peers. On an average, a peer consumer has at least 1 peer leaser available to offload in proximity all the time. While proximal peers (D2D and cloudlets) are available to offload, it is preferable that a peer consumer offloads to the cloud as the energetic source and computational power of the cloud is unlimited and scalable. Fig. 11 shows the average and median RTT of each participant to remote cloud during each hour of the day. While the mean captures outliers of high latency, the median provides insights about the availability of network connectivity. From the results, we can identify multiple levels of availability to remote cloud. Naturally, availability means that the RTT is lower enough such that the device can offload without inducing a counterproductive effect in the device. For instance, if we consider that the minimum latency required to offload is 200 ms, then, p0 has no available connectivity to offload to the cloud. Certainly, we can appreciate that every participant has network connectivity most of the time (on an average 80% of the total time). However, the latency is too high, which is unsuitable to offload. Thus, even if there is network connectivity available, it is not advisable to offload to cloud. Our results confirm that, a peer consumer has more availability to offload to other peers leasers in proximity rather than offloading to a peer in the cloud.

By merging all the offloading models into a hybrid one, a peer consumer has a larger spectrum of offloading support in a hybrid model rather than a classical one. To demonstrate this, we model the availability of peer leasers during the day for participants p0, p1, p3 and p5. Similarly with RTT, we consider that the minimum latency required to offload is 200 ms. Thus, the higher the RTT, the less available is the peer in the cloud to offload. In case of peers via D2D, the availability is measured based on the number of peers available to offload. Moreover, since the computational offloading is more valuable when the device experiences low energy levels, we include the battery levels that the peers experienced during the day. The results are shown in Fig. 12. We can observe that the availability of offloading support increases in a hybrid system up to 85% for p0, 20% for p1, 100% for p3 and 25% for p5.

Results: credit simulation—We simulate the exchange of credit in order to understand the offloading interoperability in the hybrid system. By relying on the traces of each participant (represented as p7, p8, p9), we calculate the exchange of credit on three ego networks. We rely on these three participants because they are the ones whose devices were able to discover the most among them. We select the time units for which all the three are present. For each participant, we calculate the median battery life during this time period. The average of the three medians is 68, which sets the criterion for offloading the task from one peer to other. This same threshold is used to determine whether a peer leaser will accept the offloading

Please cite this article in press as: H. Flores, et al., Social-aware hybrid mobile offloading, Pervasive and Mobile Computing (2016), http://dx.doi.org/10.1016/j.pmcj.2016.09.014

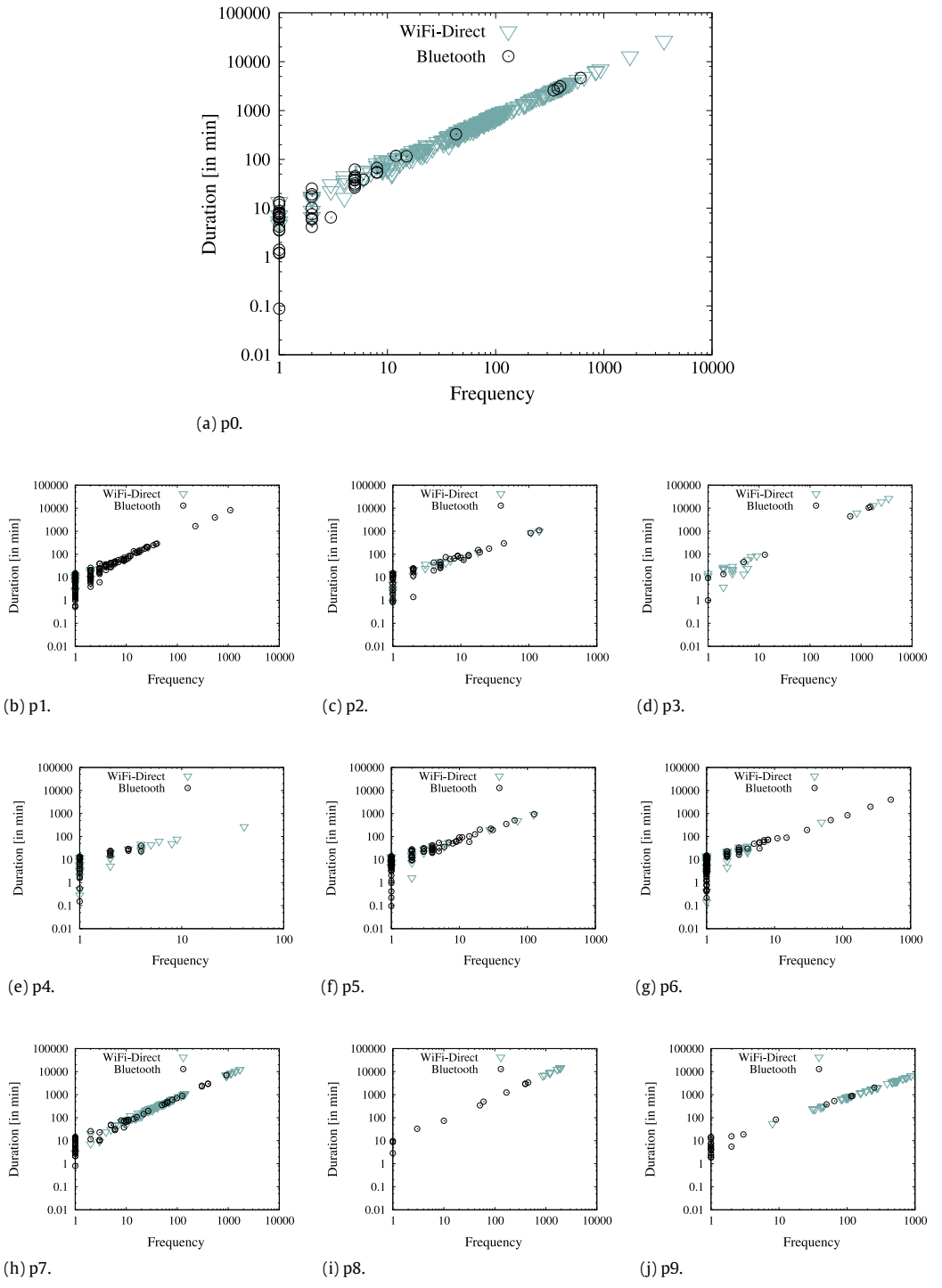
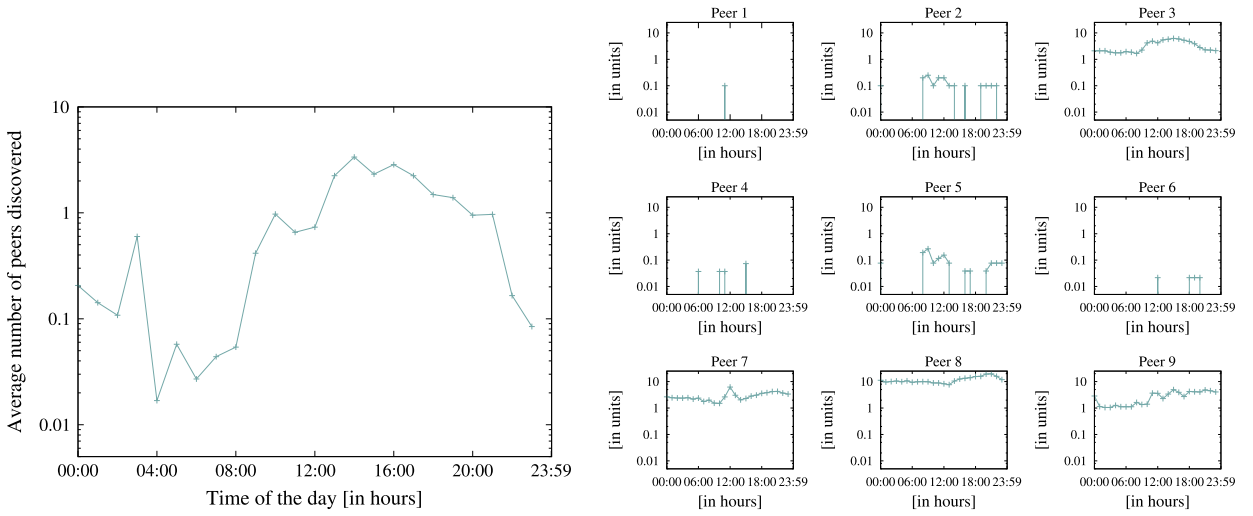


Fig. 7. Community analysis based on frequency and duration of the peers. (a)–(j) Infrastructure stability detected by each participant.

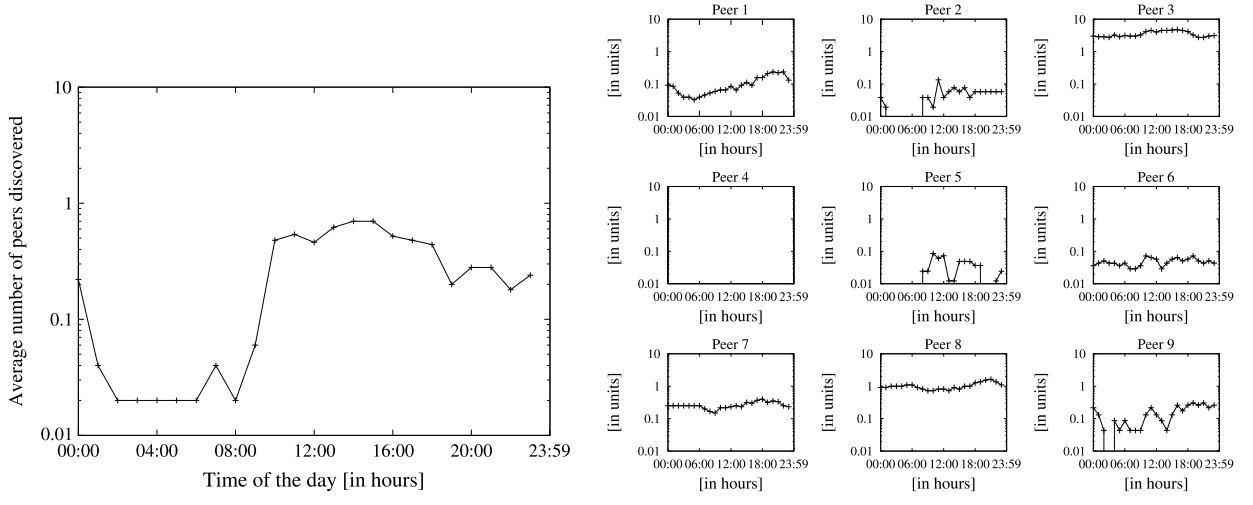
task from the peer consumer. The simulation offloading to peer in the cloud as the RTT is also available within the traces. In case the battery life is lower than 68, the participant attempts to offload the task to one of the other two users. In case the other two users have battery life less than 68, the participant offloads the task to the peer in the cloud.(represented by C in the graphs).

In each experiment, we only assumed that at a time, only a single participant wants to offload a task with the other two participants. Fig. 13(a)–(c) show the credit changes with respect to the participant p7, p8 and p9. Initially, each participant is



(a) p0. (b) p1, p2, p3, p4, p5, p6, p7, p8, p9.

Fig. 8. Average number of peers discovered via WiFi-Direct. (a) Average number of peers discovered by participant p0. (b) Average number of peers discovered by the rest of the participants.



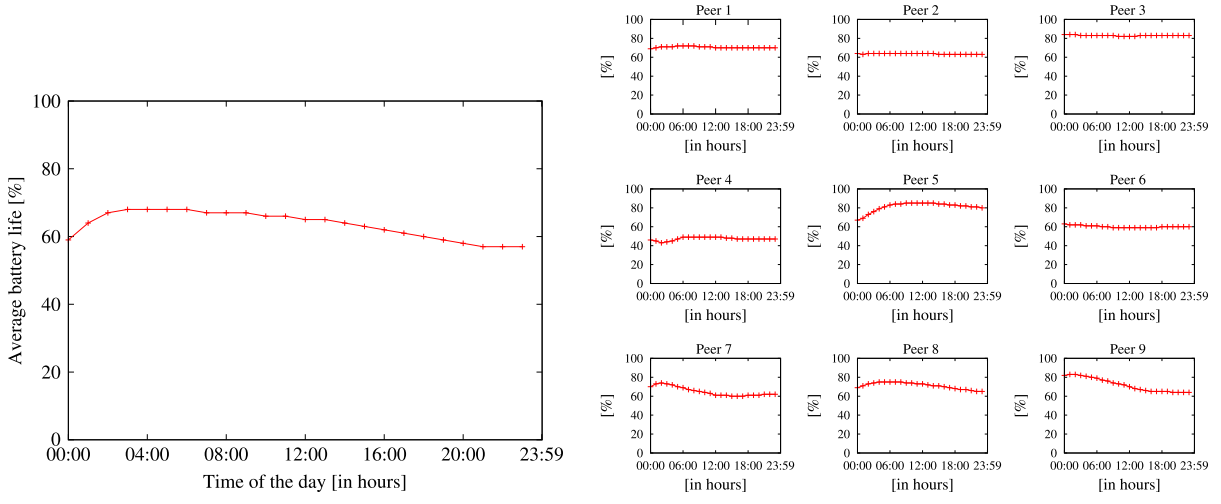
(a) p0. (b) p1, p2, p3, p4, p5, p6, p7, p8, p9.

Fig. 9. Average number of peers discovered via Bluetooth. (a) Average number of peers discovered by participant p0. (b) Average number of peers discovered by the rest of the participants.

allocated with 1000 credits. Once a participant consumes all the 1000 credits, the credits turn negative. One option could be to buy more credits. However, for simplicity, we showed the negative credits for all the three cases. In all cases, the peer in the cloud (C) is the last peer whose credits increases more than 1000 as the users first try to offload the tasks to its neighbors first. In all cases, we notice that, on an average the two participants are able to make their credits double. This shows that there is enough motivation for the users to contribute their resources and increase their credits, which can be utilize later on.

Results: cellular network (3G/3G LTE)—Since offloading can happen also using the cellular network, we evaluate the impact of using 3G and 3G LTE as mean to achieve offloading support. We rely on the dataset provided by NetRadar.[5] The dataset was collected from 2015 in the metropolitan area of Helsinki, Finland. Since the latency in the cellular network also depends on the quality of service provided by the vendor, we analyze three different mobile providers (DNA, Elisa and Sonera). Fig. 14(d) (DNA), 14(e) (Elisa) and 14(f) (Sonera) shows the number of samples that were taken to characterize the communication latency for each provider. Fig. 14(a)–(c) shows the average latency of the communication (RTT) for each provider, respectively. From the results, we can observe that the average RTT using 3G LTE for each cellular operator is

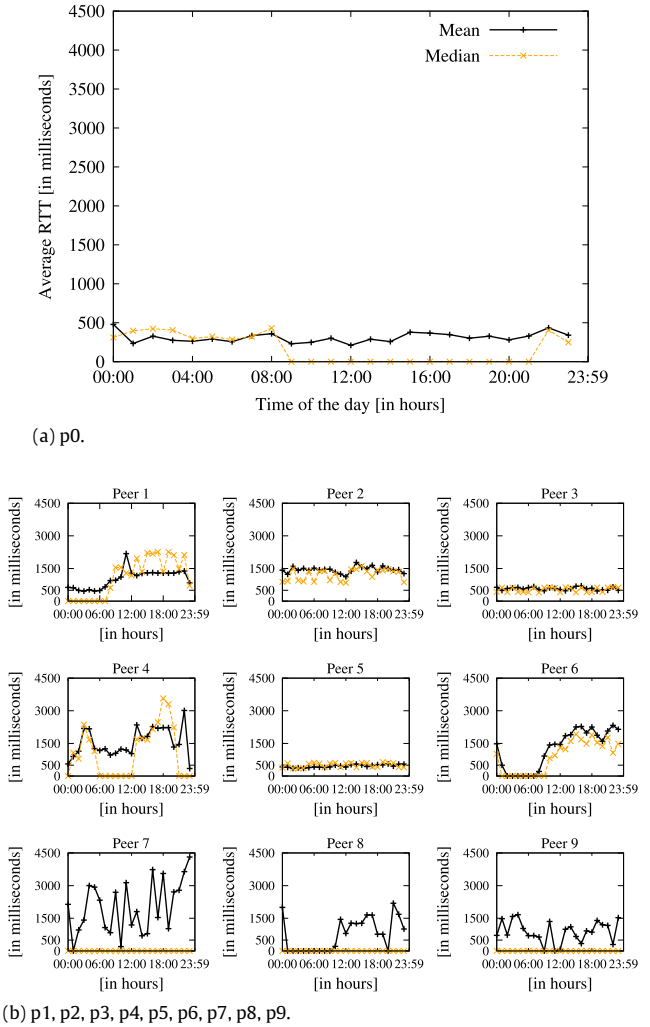
5 https://www.netradar.org/en.



(a) p0.

(b) p1, p2, p3, p4, p5, p6, p7, p8, p9.

Fig. 10. Average battery draining of the mobile users. (a) Average battery of participant p0. (b) Average battery of the rest of the participants.

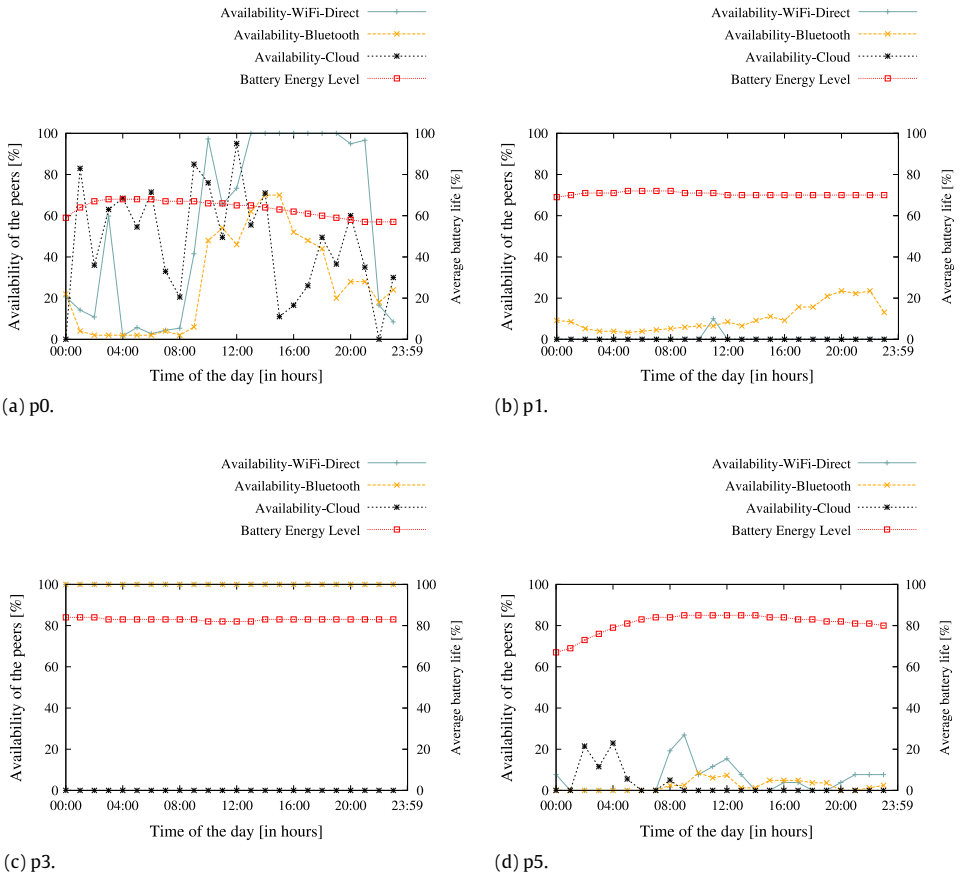


(a) p0.

(b) p1, p2, p3, p4, p5, p6, p7, p8, p9.

Fig. 11. Average and median RTT experienced by the mobile users. (a) Average and median RTT of participant p0. (b) Average and median RTT of the rest of the participants.

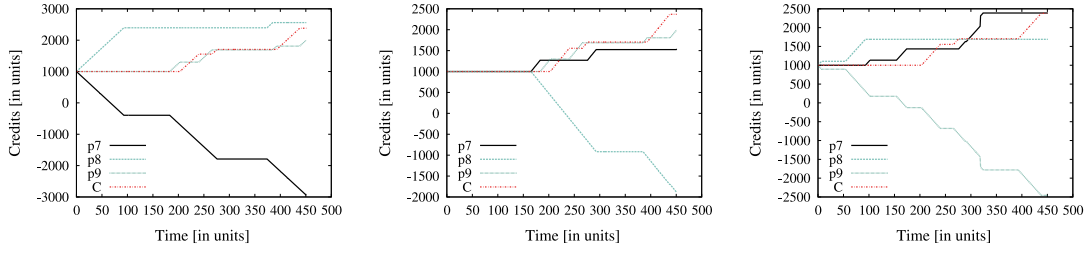
Please cite this article in press as: H. Flores, et al., Social-aware hybrid mobile offloading, Pervasive and Mobile Computing (2016), http://dx.doi.org/10.1016/j.pmcj.2016.09.014



(a) p0. (b) p1.

(c) p3. (d) p5.

Fig. 12. Daily basis availability of offloading support for (a) p0, (b) p1, (c) p3, and (d) p5.



(a) p7. (b) p8. (c) p9.

Fig. 13. Simulation of credit for (a) p7, (b) p8, and (c) p9.

≈50 ms, which is comparable with cloudlet latency. We also can observe that the average RTT using 3G for each cellular operator is ≈125 ms (DNA), ≈130 ms (Elisa) and ≈150 ms (Sonera). While there is notable difference between 3G and 3G LTE, both provide high latency communication to achieve offloading support.

6. Discussion

Based on the results of our experimental testbeds, we now discuss the advantages and limitations of *HyMobi*.

6.1. Social awareness

While our hybrid model relies on social relations based on credit to establish communication between the devices, other relations can be considered, e.g., altruism, friendship, etc. However, based on our previous work [4], we identified that while a user is willing to help his/her friends in some cases, there is not strong motivation for doing it often. From the study, it is demonstrated that the battery life of a user's device is quantifiable. Thus, the user expects some kind of reward from giving

Please cite this article in press as: H. Flores, et al., Social-aware hybrid mobile offloading, Pervasive and Mobile Computing (2016), http://dx.doi.org/10.1016/j.pmcj.2016.09.014

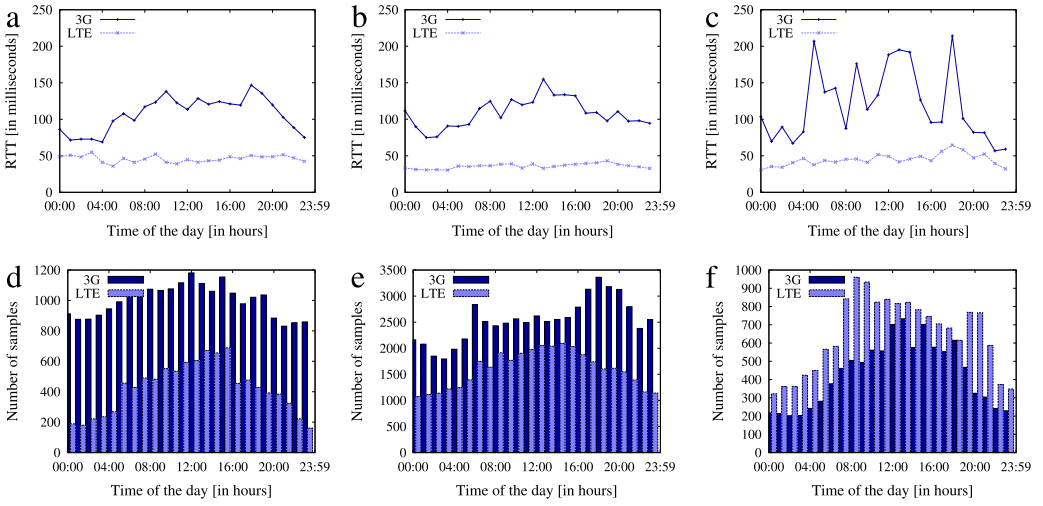


Fig. 14. Latency of the network communication by mobile operator for 3G/LTE technologies ((a) DNA, (b) Elisa and (c) Sonera), size of the sample (number of service subscribers (d) DNA, (e) Elisa, and (f) Sonera).

away its energetic resources. However, it is hard to notice from the leaser the impact of processing other's task in his/her device in a short period of time. Therefore, it is complex to charge the leasing of the mobile resources. Thus, our proposed credit mechanism provides a more realistic strategy to lease and acquire computational resources from other users. Our mechanism is more appealing for a user in practice as it allows users to accumulate credit which can be used later to aid their mobile device.

6.2. Community formation

We propose a community formation criteria that rely on the factors of stability, history of connections and reputation of peers. While these factors are enough to form communities, they are not enough to ensure that a peer is trustful. Mechanisms such as reputation and stability can be faked easily by malicious peers. However, *HyMobi* requires the explicit intervention of a user before a peer can be considered as a member of a community. Thus, in order to inject an attack into the system, first, the user needs to be tricked by the malicious peer.

On the other hand, it is a complex task to calculate the stability of a peer, mainly because the device may not be visible all the time due to privacy issues or saving energy purposes, e.g., Bluetooth is off. In fact, recently, mobile platforms, e.g., Android and iOS, are making the process of discovery more secure for the user, such that when the device is discovered, it uses each time a random address to identify itself to the peers. This introduces an extra level of complexity to estimate stability.

6.3. Cellular network

From the cellular network analysis presented in Section 5, we observe that the latency in the communication using 3G and 3G LTE technologies is comparable with D2D and cloudlet environments. Thus, by using 3G or 3G LTE, it is possible to offload to cloud without inducing a high computational effort.

However, we need to mention that the utilization of cellular network has an extra cost for the user. Moreover, it is expected that the mobile traffic will increase with the rise of the Internet of Things (IoT) paradigm. Thus, strategies of *5G, Fog and Edge Computing* will be critical to release the mobile network from the extensive data transfer. In this context, *HyMobi* is clearly a framework designed to foster *fog offloading*.

7. Conclusions and future directions

Offloading is an important strategy for the next generation of smart infrastructures that aim to augment the constrained capabilities of a mobile device to enhance the Quality-of-Service (QoS) and Quality-of-Experience (QoE) of end users. Computational offloading has been proposed as a solution for saving the battery life of the mobile devices. However, in practice, acquiring offloading support is rather complicate to achieve for a mobile device due to many reasons, e.g., unavailable network, high latency, cloud cost, cellular network cost, etc. Thus, in this paper, we proposed a hybrid system for computational offloading which increases the spectrum of opportunities to offload by exploiting transient infrastructure. By merging cloudlet, remote cloud and D2D communication, we found that it is possible to improve the availability of offloading support for mobile users.

Please cite this article in press as: H. Flores, et al., Social-aware hybrid mobile offloading, Pervasive and Mobile Computing (2016), http://dx.doi.org/10.1016/j.pmcj.2016.09.014

Naturally, the deployment and design of a hybrid system require to overcome many unification challenges. Thus, we develop a credit and reputation mechanism which exploits social aspects in order to sustain our hybrid system. The ultimate goal of our system is to create communities, in which a device can lease and acquire offloading support based on incentives. Lastly, our proposed system gives a step forward to achieve a *fog computing* architecture for mobile offloading. We provide our framework, use cases and tools as open source in GitHub.

In our current system, the community assignment has been made based on stability, which is calculated based on the frequency and duration in which a device is detected. Since this requires to activate the discovery interfaces of the mobile, e.g., Bluetooth, WiFi-Direct, we would like to incorporate a community prediction approach in our system. The approach can learn based on past users' request experiences and propose an efficient solution for future incoming requests. This prediction approach would help to improve the discovery process of the super-peer and save its energy. In this context, the super-peer could infer what devices are available at a particular location rather than monitoring the available peers using a Bluetooth or WiFi-Direct discovery processes, which are energy draining for the mobile device. We are also interested on exploring techniques that can accelerate the processing of a task by distributing the processing of its execution across multiple devices (mobile code parallelization).

Acknowledgments

The authors thank the anonymous reviewers for their insightful comments. This work is partially funded by the Academy of Finland (Grants 276786-AWARE, 285062-iCYCLE, 286386-CPDSS, 285459-iSCIENCE), and the European Commission (Grants PCIG11-GA-2012-322138 and 645706-GRAGE).

References

[1] D. Ferreira, A.K. Dey, V. Kostakos, Understanding human-smartphone concerns: a study of battery life, in: International Conference on Pervasive Computing, Pervasive, Berlin, Heidelberg, 2011.
[2] P. Ferreira, M. McGregor, A. Lampinen, Caring for batteries: Maintaining infrastructures and mobile social contexts, in: 17th International Conference on Human-Computer Interaction with Mobile Devices and Services, ACM, MobileHCI'15, 2015.
[3] F.F.-H. Nah, K. Siau, H. Sheng, The value of mobile applications: a utility company study, Commun. ACM 48 (2) (2005) 85–90.
[4] S. Hosio, D. Ferreira, J. Goncalves, N. van Berkel, C. Luo, M. Ahmed, H. Flores, V. Kostakos, Monetary assessment of battery life on smartphones, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI, 2016.
[5] H. Flores, P. Hui, S. Tarkoma, Y. Li, S. Srirama, R. Buyya, Mobile code offloading: from concept to practice and beyond, IEEE Commun. Mag. 53 (3) (2015) 80–88.
[6] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, H.-I. Yang, The case for cyber foraging, in: Proceedings of the 10th Workshop on ACM SIGOPS European Workshop, ACM, 2002.
[7] B. Han, P. Hui, V.A. Kumar, M.V. Marathe, J. Shao, A. Srinivasan, Mobile data offloading through opportunistic communications and social participation, IEEE Trans. Mob. Comput. 11 (5) (2012) 821–834.
[8] K. Kumar, Y.-H. Lu, Cloud computing for mobile users: Can offloading computation save energy? Computer 43 (4) (2010) 51–56.
[9] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, Maui: making smartphones last longer with code offload, in: 8th International Conference on Mobile Systems, Applications, and Services, ACM, 2010.
[10] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti, Clonecloud: elastic execution between mobile device and cloud, in: 6th Conference on Computer Systems, 2011.
[11] S. Kosta, A. Aucinas, P. Hui, R. Mortier, X. Zhang, Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading, in: IEEE INFOCOM, 2012.
[12] H. Flores, S. Srirama, Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning, in: 4th ACM MobiSys Workshop on Mobile Cloud Computing and Services, 2013.
[13] M.S. Gordon, D.A. Jamshidi, S. Mahlke, Z.M. Mao, X. Chen, Comet: code offload by migrating execution transparently, in: 10th Conference on Operating Systems Design and Implementation, USENIX, 2012.
[14] C. Shi, K. Habak, P. Pandurangan, M. Ammar, M. Naik, E. Zegura, Cosmos: Computation offloading as a service for mobile devices, MobiHoc 2014.
[15] H. Flores, S. Srirama, Mobile code offloading: should it be a local decision or global inference?, in: ACM 11th Annual International Conference on Mobile Systems, Applications, and Services, 2013.
[16] M. Satyanarayanan, P. Bahl, R. Caceres, N. Davies, The case for vm-based cloudlets in mobile computing, IEEE Pervasive Comput. 8 (4) (2009) 14–23.
[17] C. Shi, V. Lakafosis, M.H. Ammar, E.W. Zegura, Serendipity: enabling remote computing among intermittently connected mobile devices, in: ACM 13th International Symposium on Mobile Ad Hoc Networking and Computing, 2012.
[18] H. Flores, Service-oriented and evidence-aware mobile cloud computing (Ph.D. thesis), 2015.
[19] A. Passarella, M. Conti, Analysis of individual pair and aggregate intercontact times in heterogeneous opportunistic networks, IEEE Trans. Mob. Comput. 12 (12) (2013) 2483–2495.
[20] C. Boldrini, M. Conti, A. Passarella, The stability region of the delay in pareto opportunistic networks, IEEE Trans. Mob. Comput. 14 (1) (2015) 180–193.
[21] T. Verbelen, P. Simoens, F. De Turck, B. Dhoedt, Cloudlets: bringing the cloud to the mobile user, in: 3rd Mobisys Workshop on Mobile Cloud Computing and Services, ACM, 2012.
[22] H. Flores, S.N. Srirama, Mobile cloud middleware, J. Syst. Softw. 92 (2014) 82–94.
[23] H. Flores, S.N. Srirama, C. Paniagua, A generic middleware framework for handling process intensive hybrid cloud services from mobiles, in: Proceedings of the 9th International Conference on Advances in Mobile Computing and Multimedia, ACM, 2011, pp. 87–94.
[24] F. Xia, F. Ding, J. Li, X. Kong, L.T. Yang, J. Ma, Phone2cloud: Exploiting computation offloading for energy saving on smartphones in mobile cloud computing, Inf. Syst. Front. 16 (1) (2014) 95–111.
[25] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, R. Govindan, Odessa: enabling interactive perception applications on mobile devices, in: ACM 9th International Conference on Mobile Systems, Applications, and Services, 2011.
[26] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, D. Milojicic, Adaptive offloading for pervasive computing, IEEE Pervasive Comput. 3 (3) (2004) 66–73.
[27] Y. Li, T. Wu, P. Hui, D. Jin, S. Chen, Social-aware d2d communications: qualitative insights and quantitative analysis, IEEE Commun. Mag. 52 (6) (2014) 150–158.
[28] J. Mass, S.N. Srirama, H. Flores, C. Chang, Proximal and social-aware device-to-device communication via audio detection on cloud, in: 13th International Conference on Mobile and Ubiquitous Multimedia, 2014.
[29] K. Habak, M. Ammar, K.A. Harras, E. Zegura, Femtoclouds: Leveraging mobile devices to provide cloud service at the edge, in: IEEE International Conference on Cloud Computing, 2015.

Please cite this article in press as: H. Flores, et al., Social-aware hybrid mobile offloading, Pervasive and Mobile Computing (2016), http://dx.doi.org/10.1016/j.pmcj.2016.09.014

[30] A. Mtibaa, K.A. Harras, K. Habak, M. Ammar, E.W. Zegura, Towards mobile opportunistic computing, in: IEEE 8th International Conference on Cloud Computing, 2015.
[31] A. Mtibaa, K.A. Harras, A. Fahim, Towards computational offloading in mobile device clouds, in: IEEE 5th International Conference on Cloud Computing Technology and Science, CloudCom, 2013.
[32] C. Shi, M.H. Ammar, E.W. Zegura, M. Naik, Computing in cirrus clouds: the challenge of intermittent connectivity, in: Proceedings of the 1st MCC Workshop on Mobile Cloud Computing, 2012.
[33] A. Noori, D. Giustiniano, Hycloud: a hybrid approach toward offloading cellular content through opportunistic communication, in: 11th Annual International Conference on Mobile Systems, Applications, and Services, 2013.
[34] L. Buttyán, J.-P. Hubaux, Stimulating cooperation in self-organizing mobile ad hoc networks, Mob. Netw. Appl. 8 (5) (2003) 579–592.
[35] T. Fahad, R.J. Askwith, A node misbehaviour detection mechanism for mobile ad-hoc networks, in: The 7th Annual PostGraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting, 2006.
[36] D. Levin, Punishment in selfish wireless networks: A game theoretic analysis., in: NetEcon, 2006.
[37] D. Myers, Self and selfishness in online social play, in: Digital Games Research Association, DiGRA, 2007.
[38] E. Koutsoupias, C. Papadimitriou, Worst-case equilibria, in: 16th Symposium on Theoretical Aspects of Computer Science, 1999.
[39] T. Moscibroda, S. Schmid, R. Wattenhofer, On the topologies formed by selfish peers, in: 5th International Workshop on Peer-to-Peer Systems, IPTPS, 2006.
[40] A. Datta, S. Buchegger, L.-H. Vu, T. Strufe, K. Rzadca, Decentralized online social networks, 2010.
[41] O.F. Gonzalez, M. Howarth, G. Pavlou, Detection of packet forwarding misbehavior in mobile ad-hoc networks, in: Proceedings of the 5th International Conference on Wired/Wireless Internet Communications, WWIC, 2007.
[42] K. Balakrishnan, J. Deng, P. Varshney, Twoack: preventing selfishness in mobile ad hoc networks, in: IEEE Wireless Communications and Networking Conference, 2005.
[43] D. Feng, Y. Zhu, X. Luo, Cooperative incentive mechanism based on game theory in manet, in: International Conference on Networking and Digital Society, ICNDS, 2009.
[44] Q. He, D. Wu, P. Khosla, Sori: a secure and objective reputation-based incentive scheme for ad-hoc networks, in: IEEE Conference on Wireless Communications and Networking, WCNC, 2004.
[45] F. Martignon, S. Paris, A. Capone, A framework for detecting selfish misbehavior in wireless mesh community networks, in: Proceedings of the 5th ACM Symposium on QoS and Security for Wireless and Mobile Networks, Q2SWinet, 2009.
[46] F. Milan, J.J. Jaramillo, R. Srikant, Achieving cooperation in multihop wireless networks of selfish nodes, in: Workshop on Game Theory for Communications and Networks, GameNets, 2006.
[47] Y. Li, G. Su, D.O. Wu, D. Jin, L. Su, L. Zeng, The impact of node selfishness on multicasting in delay tolerant networks, IEEE Trans. Veh. Technol. 60 (5) (2011) 2224–2238.
[48] R. Ma, S. Lee, J. Lui, D. Yau, An incentive mechanism for p2p networks, in: 24th International Conference on Distributed Computing Systems, 2004.
[49] B. Cohen, Incentives build robustness in bittorrent, in: IPTPS, 2003.
[50] M. Babaioff, J. Chuang, M. Feldman, Incentives in peer-to-peer systems, in: Algorithmic Game Theory. Cambridge, 2007, pp. 593–611.
[51] R. Sharma, A. Datta, M.D. Amico, P. Michiardi, An empirical study of availability in friend-to-friend storage systems, in: IEEE International Conference on Peer-to-Peer Computing, P2P, 2011.
[52] K. Rzadca, A. Datta, S. Buchegger, Replica placement in p2p storage: Complexity and game theoretic analyses, in: IEEE 30th International Conference on Distributed Computing Systems, ICDCS, 2010.
[53] R. Sharma, A. Datta, Supernova: Super-peers based architecture for decentralized online social networks, in: 4th International Conference on Communication Systems and Networks, COMSNETS, 2012.
[54] L.G. Jaimes, I. Vergara-Laurens, M.A. Labrador, A location-based incentive mechanism for participatory sensing systems with budget constraints, in: IEEE International Conference on Pervasive Computing and Communications, PerCom, 2012.
[55] T. Ning, Z. Yang, X. Xie, H. Wu, Incentive-aware data dissemination in delay-tolerant mobile networks, in: 8th Conference on Sensor, Mesh and Ad Hoc Communications and Networks, SECON, 2011.
[56] S. Zhong, J. Chen, Y. Yang, Sprite: a simple, cheat-proof, credit-based system for mobile ad-hoc networks, in: 22th Annual Joint Conference of the IEEE Computer and Communications, INFOCOM, 2003.
[57] H. Flores, S.N. Srirama, Mobile cloud messaging supported by xmpp primitives, in: Proceedings of the Fourth ACM Workshop on Mobile Cloud Computing and Services, (MCS 2013), ACM, 2013.

Please cite this article in press as: H. Flores, et al., Social-aware hybrid mobile offloading, Pervasive and Mobile Computing (2016), http://dx.doi.org/10.1016/j.pmcj.2016.09.014